Week 2

Chapter 1 - Pandas

Pandas

- Pandas is an open-source library that allows to you perform data manipulation and analysis in Python.
- Pandas Python library offers data manipulation and data operations for numerical tables and time series.
- Pandas provide an easy way to create, manipulate, and wrangle the data

- ### **Install Libraries**

In [ ]:
```
# pip install pandas
# pip install numpy
```

- #### **Import Libraries**

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
# Object Creation
pd.Series([1,3,np.nan,5,7,9])
```

Out[ ]:
```
0    1.0
1    3.0
2    NaN
3    5.0
4    7.0
5    9.0
dtype: float64
```

In [ ]:
```
# ranging "2022-01-01 to 2022-01-13" in a variable date
date = pd.date_range("20220101",periods=13)
date
```

Out[ ]:
```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:
```
# random integers having 6 rows and 4 columns
np.random.randn(6,4)
```

Out[ ]:
```
array([[ 0.0699486 ,  0.32640312,  1.18180645, -0.60114338],
       [ 1.13070046,  0.04541895,  0.77859725,  0.2037714 ],
       [-0.69499029, -1.19852794, -0.64524166,  0.12637739],
       [ 0.40919921, -0.0815135 , -1.5768496 , -1.62732821],
       [ 0.33512639, -0.32343125, -0.7523412 ,  1.34194579],
       [ 2.00636812, -1.79655335,  0.51041365, -0.27552775]])
```

In [ ]:
```
df = pd.DataFrame(np.random.randn(13,4),index=date,columns=list('ABCD'))
df
```

Out[ ]:

	A	B	C	D
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-04	-1.921905	0.609435	-1.287698	1.656692
2022-01-05	0.019881	-0.093638	0.125498	0.698151
2022-01-06	0.319068	0.773514	-1.711611	-1.670382
2022-01-07	0.513668	0.918490	-1.409920	1.736568
2022-01-08	0.698414	-0.082506	-2.120228	-0.828376
2022-01-09	-0.747636	0.176316	-1.137482	0.855713
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-11	0.247761	0.743186	-2.009937	0.861352
2022-01-12	-0.342993	0.389217	-0.468365	0.442806
2022-01-13	0.317621	-0.626462	0.787241	0.271840

In [ ]:
```
# showing first 5 rows
df.head()
```

Out[ ]:

	A	B	C	D
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-04	-1.921905	0.609435	-1.287698	1.656692
2022-01-05	0.019881	-0.093638	0.125498	0.698151

In [ ]:
```
# showing last 5 rows
df.tail()
```

Out[ ]:

	A	B	C	D
2022-01-09	-0.747636	0.176316	-1.137482	0.855713
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-11	0.247761	0.743186	-2.009937	0.861352
2022-01-12	-0.342993	0.389217	-0.468365	0.442806
2022-01-13	0.317621	-0.626462	0.787241	0.271840

In [ ]:
```
# showing index of dataframe
df.index
```

Out[ ]:
```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:
```
# showing columns of dataframe
df.columns
```

Out[ ]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [ ]:
```
# converting dataframe into numpy array
```

```python
df.to_numpy()
```

Out[ ]:
```
array([[ 1.50394486,  0.52714022,  0.93148586, -1.17064181],
       [ 2.66606759, -0.67432053, -0.71537134,  0.09261027],
       [ 0.5439125 , -0.22438701, -0.56456501,  0.61797971],
       [-1.92190526,  0.60943492, -1.28769776,  1.65669198],
       [ 0.01988134, -0.09363808,  0.12549784,  0.69815124],
       [ 0.31906838,  0.77351413, -1.71161091, -1.67038199],
       [ 0.51366775,  0.91848961, -1.40992004,  1.73656766],
       [ 0.69841442, -0.08250585, -2.12022817, -0.82837599],
       [-0.74763607,  0.17631627, -1.13748242,  0.85571326],
       [ 0.41258227,  0.31740479,  0.97425947,  0.94465307],
       [ 0.24776079,  0.74318619, -2.00993732,  0.86135181],
       [-0.34299334,  0.38921685, -0.4683654 ,  0.44280604],
       [ 0.3176208 , -0.62646228,  0.787241  ,  0.27183955]])
```

In [ ]:
```python
# showing statistics
df.describe()
```

Out[ ]:

	A	B	C	D
count	13.000000	13.000000	13.000000	13.000000
mean	0.325414	0.211799	-0.662053	0.346843
std	1.074152	0.520961	1.088150	1.022969
min	-1.921905	-0.674321	-2.120228	-1.670382
25%	0.019881	-0.093638	-1.409920	0.092610
50%	0.319068	0.317405	-0.715371	0.617980
75%	0.543912	0.609435	0.125498	0.861352
max	2.666068	0.918490	0.974259	1.736568

In [ ]:
```python
# building new dataframe df2
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3]*4, dtype="int32"),
        "E": pd.Categorical(["test","train","test","train"]),
        "F": "foo"
```

```
    }
)
df2
```

Out[ ]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [ ]:

```
# dimensional types eg integer,float
df2.dtypes
```

Out[ ]:

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

In [ ]:

```
# Transpose
df2.T
```

Out[ ]:

	0	1	2	3
A	1.0	1.0	1.0	1.0
B	2013-01-02 00:00:00	2013-01-02 00:00:00	2013-01-02 00:00:00	2013-01-02 00:00:00
C	1.0	1.0	1.0	1.0
D	3	3	3	3
E	test	train	test	train
F	foo	foo	foo	foo

In [ ]:

```
# sort by index
df.sort_index(axis=0,ascending=False)
```

Out[ ]:

	A	B	C	D
2022-01-13	0.317621	-0.626462	0.787241	0.271840
2022-01-12	-0.342993	0.389217	-0.468365	0.442806
2022-01-11	0.247761	0.743186	-2.009937	0.861352
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-09	-0.747636	0.176316	-1.137482	0.855713
2022-01-08	0.698414	-0.082506	-2.120228	-0.828376
2022-01-07	0.513668	0.918490	-1.409920	1.736568
2022-01-06	0.319068	0.773514	-1.711611	-1.670382
2022-01-05	0.019881	-0.093638	0.125498	0.698151
2022-01-04	-1.921905	0.609435	-1.287698	1.656692
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-01	1.503945	0.527140	0.931486	-1.170642

In [ ]:

```
# Sort Values
df.sort_values(by = 'C',axis=0, ascending=False)
```

Out[ ]:

	A	B	C	D
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-13	0.317621	-0.626462	0.787241	0.271840
2022-01-05	0.019881	-0.093638	0.125498	0.698151
2022-01-12	-0.342993	0.389217	-0.468365	0.442806
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-09	-0.747636	0.176316	-1.137482	0.855713
2022-01-04	-1.921905	0.609435	-1.287698	1.656692
2022-01-07	0.513668	0.918490	-1.409920	1.736568
2022-01-06	0.319068	0.773514	-1.711611	-1.670382
2022-01-11	0.247761	0.743186	-2.009937	0.861352
2022-01-08	0.698414	-0.082506	-2.120228	-0.828376

In [ ]:

```
# Indexing
# row wise selection
df[0:6]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-04	-1.921905	0.609435	-1.287698	1.656692
2022-01-05	0.019881	-0.093638	0.125498	0.698151
2022-01-06	0.319068	0.773514	-1.711611	-1.670382

In [ ]:

```
# cross section data
df.loc[date[0]]
```

Out[ ]:
```
A    1.503945
B    0.527140
C    0.931486
D   -1.170642
Name: 2022-01-01 00:00:00, dtype: float64
```

In [ ]:
```
# having columns 'A' and 'B' in dataframe
df.loc[:,["A","B"]]
```

Out[ ]:

	A	B
2022-01-01	1.503945	0.527140
2022-01-02	2.666068	-0.674321
2022-01-03	0.543912	-0.224387
2022-01-04	-1.921905	0.609435
2022-01-05	0.019881	-0.093638
2022-01-06	0.319068	0.773514
2022-01-07	0.513668	0.918490
2022-01-08	0.698414	-0.082506
2022-01-09	-0.747636	0.176316
2022-01-10	0.412582	0.317405
2022-01-11	0.247761	0.743186
2022-01-12	-0.342993	0.389217
2022-01-13	0.317621	-0.626462

In [ ]:
```
# indexing
df.loc["20220101":"20220103",["A","B",'C']]
```

Out[ ]:

	A	B	C
2022-01-01	1.503945	0.527140	0.931486
2022-01-02	2.666068	-0.674321	-0.715371
2022-01-03	0.543912	-0.224387	-0.564565

In [ ]:
```
df.loc[["20220101","20220103"],["A","B",'C']]
```

Out[ ]:

	A	B	C
2022-01-01	1.503945	0.527140	0.931486
2022-01-03	0.543912	-0.224387	-0.564565

In [ ]:
```
# at 0 index and 'A' column
df.at[date[0],'A']
```

Out[ ]:
```
1.5039448630502772
```

In [ ]:
```
# index of location
df.iloc[0,:]
```

Out[ ]:
```
A    1.503945
B    0.527140
C    0.931486
D   -1.170642
Name: 2022-01-01 00:00:00, dtype: float64
```

In [ ]:
```
# selection or filterataion
df[df["A"]>0]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-02	2.666068	-0.674321	-0.715371	0.092610
2022-01-03	0.543912	-0.224387	-0.564565	0.617980
2022-01-05	0.019881	-0.093638	0.125498	0.698151
2022-01-06	0.319068	0.773514	-1.711611	-1.670382
2022-01-07	0.513668	0.918490	-1.409920	1.736568
2022-01-08	0.698414	-0.082506	-2.120228	-0.828376
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-11	0.247761	0.743186	-2.009937	0.861352
2022-01-13	0.317621	-0.626462	0.787241	0.271840

In [ ]:

```
# copying and saving df into df1
df1 = df.copy()
# adding another column "E" in dataframe df1
df1["E"] = ["One","Two","One","Four","Three","Three","One","Two","One","Four","Three","Three","Five"]
# viewing first 5 rows
df1.head()
```

Out[ ]:

	A	B	C	D	E
2022-01-01	1.503945	0.527140	0.931486	-1.170642	One
2022-01-02	2.666068	-0.674321	-0.715371	0.092610	Two
2022-01-03	0.543912	-0.224387	-0.564565	0.617980	One
2022-01-04	-1.921905	0.609435	-1.287698	1.656692	Four
2022-01-05	0.019881	-0.093638	0.125498	0.698151	Three

Assignment 1

In [ ]:

```
#  Multi Condition and Filtering
df[(df["A"]>0) & (df["B"]>0)]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.503945	0.527140	0.931486	-1.170642
2022-01-06	0.319068	0.773514	-1.711611	-1.670382
2022-01-07	0.513668	0.918490	-1.409920	1.736568
2022-01-10	0.412582	0.317405	0.974259	0.944653
2022-01-11	0.247761	0.743186	-2.009937	0.861352

Assignment 2

In [ ]:

```
df.sum(axis=1)
```

Out[ ]:

```
2022-01-01    1.791929
2022-01-02    1.368986
2022-01-03    0.372940
2022-01-04   -0.943476
2022-01-05    0.749892
2022-01-06   -2.289410
2022-01-07    1.758805
2022-01-08   -2.332696
2022-01-09   -0.853089
2022-01-10    2.648900
2022-01-11   -0.157639
2022-01-12    0.020664
2022-01-13    0.750239
Freq: D, dtype: float64
```

In [ ]:

```
# adding another column "Mean" which is equal to average (sum of all columns divided by 4)
df1['Mean'] = df.sum(axis=1)/4
```

In [ ]:

```
# viewing first five rows
df1.head()
```

Out[ ]:

	A	B	C	D	E	Mean
2022-01-01	1.503945	0.527140	0.931486	-1.170642	One	0.447982
2022-01-02	2.666068	-0.674321	-0.715371	0.092610	Two	0.342246
2022-01-03	0.543912	-0.224387	-0.564565	0.617980	One	0.093235
2022-01-04	-1.921905	0.609435	-1.287698	1.656692	Four	-0.235869
2022-01-05	0.019881	-0.093638	0.125498	0.698151	Three	0.187473

Chapter 2 - Pandas Case Study

- ### **Exploring titanic dataset**

In [ ]:

```
# Import Libraries
import pandas as pd
import numpy as np
import seaborn as sns
```

In [ ]:

```
# loading titanic dataset from seaborn library and save in variable kashti
kashti = sns.load_dataset('titanic')
kashti.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [ ]:

```
# saving dataframe into csv file
kashti.to_csv('kashti1.csv',index=False)
```

In [ ]:

```
# understanding continuous data
```

```
kashti.describe()
```

In [ ]:
```
# droping ['deck','embark_town','embarked'] columns from kashti dataset and viewing first 5 rows
kashti.drop(['deck','embark_town','embarked'],axis=1).head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	class	who	adult_male	alive	alone
0	0	3	male	22.0	1	0	7.2500	Third	man	True	no	False
1	1	1	female	38.0	1	0	71.2833	First	woman	False	yes	False
2	1	3	female	26.0	0	0	7.9250	Third	woman	False	yes	True
3	1	1	female	35.0	1	0	53.1000	First	woman	False	yes	False
4	0	3	male	35.0	0	0	8.0500	Third	man	True	no	True

In [ ]:
```
# mean of columns of kashti dataset
kashti.mean()
```

```
C:\Users\Abdullah Cheema\AppData\Local\Temp\ipykernel_3504\936063474.py:2: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Selec
t only valid columns before calling the reduction.
  kashti.mean()
```

Out[ ]:
```
survived       0.383838
pclass         2.308642
age           29.699118
sibsp          0.523008
parch          0.381594
fare          32.204208
adult_male     0.602694
alone          0.602694
dtype: float64
```

In [ ]:
```
# counting unique values in survived column
kashti.value_counts(['survived'])
```

Out[ ]:
```
survived
0           549
1           342
dtype: int64
```

groupby()

- Pandas **groupby()** is used for grouping the data according to the categories and apply a function to the categories.

- It also helps to aggregate data efficiently.
- Pandas dataframe. **groupby()** function is used to split the data into groups based on some criteria.

In [ ]:
```
kashti.groupby(['sex','pclass']).mean()
```

Out[ ]:

sex	pclass	survived	age	sibsp	parch	fare	adult_male	alone
female	**1**	0.968085	34.611765	0.553191	0.457447	106.125798	0.000000	0.361702
	2	0.921053	28.722973	0.486842	0.605263	21.970121	0.000000	0.421053
	3	0.500000	21.750000	0.895833	0.798611	16.118810	0.000000	0.416667
male	**1**	0.368852	41.281386	0.311475	0.278689	67.226127	0.975410	0.614754
	2	0.157407	30.740707	0.342593	0.222222	19.741782	0.916667	0.666667
	3	0.135447	26.507589	0.498559	0.224784	12.661633	0.919308	0.760807

In [ ]:
```
# children (age less than 18)
kashti[kashti['age']<18].head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	Cherbourg	yes	False
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False
14	0	3	female	14.0	0	0	7.8542	S	Third	child	False	NaN	Southampton	no	True
16	0	3	male	2.0	4	1	29.1250	Q	Third	child	False	NaN	Queenstown	no	False

In [ ]:
```
# children mean
kashti[kashti['age']<18].mean()
```

```
C:\Users\Abdullah Cheema\AppData\Local\Temp\ipykernel_3504\3495249678.py:2: FutureWarning: Dropping of nuisance colu
mns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  kashti[kashti['age']<18].mean()
```

Out[ ]:

```
survived       0.539823
pclass         2.584071
age            9.041327
sibsp          1.460177
parch          1.053097
fare          31.220798
adult_male     0.159292
alone          0.203540
dtype: float64
```

In [ ]:

```
# children mean groupy
kashti[kashti['age']<18].groupby(['sex','pclass']).mean()
```

Out[ ]:

		survived	age	sibsp	parch	fare	adult_male	alone
sex	**pclass**							
female	**1**	0.875000	14.125000	0.500000	0.875000	104.083337	0.000000	0.125000
	2	1.000000	8.333333	0.583333	1.083333	26.241667	0.000000	0.166667
	3	0.542857	8.428571	1.571429	1.057143	18.727977	0.000000	0.228571
male	**1**	1.000000	8.230000	0.500000	2.000000	116.072900	0.250000	0.000000
	2	0.818182	4.757273	0.727273	1.000000	25.659473	0.181818	0.181818
	3	0.232558	9.963256	2.069767	1.000000	22.752523	0.348837	0.232558

Assignment

In [ ]:

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:

```
# loading titanic dataset into ks variable
ks = sns.load_dataset('titanic')
# viewing first five rows
ks.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [ ]:

```
# printing unique values in all columns of ks
for i in ks.columns:
    print(ks[i].unique())
```

```
[0 1]
[3 1 2]
['male' 'female']
[22.   38.   26.   35.     nan 54.    2.   27.   14.    4.   58.   20.
 39.   55.   31.   34.   15.   28.    8.   19.   40.   66.   42.   21.
 18.    3.    7.   49.   29.   65.   28.5   5.   11.   45.   17.   32.
 16.   25.    0.83 30.   33.   23.   24.   46.   59.   71.   37.   47.
 14.5  70.5  32.5  12.    9.   36.5  51.   55.5  40.5  44.    1.   61.
 56.   50.   36.   45.5  20.5  62.   41.   52.   63.   23.5   0.92 43.
 60.   10.   64.   13.   48.    0.75 53.   57.   80.   70.   24.5   6.
  0.67 30.5   0.42 34.5  74.  ]
[1 0 3 4 2 5 8]
[0 1 2 5 3 4 6]
[  7.25    71.2833   7.925   53.1      8.05     8.4583  51.8625  21.075
  11.1333  30.0708  16.7     26.55    31.275    7.8542  16.      29.125
  13.      18.       7.225   26.       8.0292  35.5     31.3875 263.
   7.8792   7.8958  27.7208 146.5208   7.75    10.5     82.1708  52.
   7.2292  11.2417   9.475   21.      41.5792  15.5     21.6792  17.8
  39.6875   7.8     76.7292  61.9792  27.75    46.9     80.      83.475
  27.9     15.2458   8.1583   8.6625  73.5     14.4542  56.4958   7.65
  29.      12.475    9.       9.5      7.7875  47.1     15.85    34.375
  61.175   20.575   34.6542  63.3583  23.      77.2875   8.6542   7.775
  24.15     9.825   14.4583 247.5208   7.1417  22.3583   6.975    7.05
  14.5     15.0458  26.2833   9.2167  79.2      6.75    11.5     36.75
   7.7958  12.525   66.6      7.3125  61.3792   7.7333  69.55    16.1
  15.75    20.525   55.      25.925   33.5     30.6958  25.4667  28.7125
   0.      15.05    39.      22.025   50.      8.4042   6.4958  10.4625
  18.7875  31.     113.275   27.      76.2917  90.       9.35    13.5
   7.55    26.25    12.275    7.125   52.5542  20.2125  86.5    512.3292
  79.65   153.4625 135.6333  19.5     29.7     77.9583  20.25    78.85
  91.0792  12.875    8.85   151.55    30.5     23.25    12.35   110.8833
 108.9     24.      56.9292  83.1583 262.375   14.     164.8667 134.5
   6.2375  57.9792  28.5    133.65    15.9      9.225   35.      75.25
  69.3     55.4417 211.5      4.0125 227.525   15.7417   7.7292  12.
 120.      12.65    18.75     6.8583  32.5      7.875   14.4     55.9
   8.1125  81.8583  19.2583  19.9667  89.1042  38.5      7.725   13.7917
   9.8375   7.0458   7.5208  12.2875   9.5875  49.5042  78.2667  15.1
   7.6292  22.525   26.2875  59.4      7.4958  34.0208  93.5    221.7792
 106.425   49.5     71.      13.8625   7.8292  39.6     17.4     51.4792
  26.3875  30.      40.125    8.7125  15.      33.      42.4     15.55
  65.      32.3208   7.0542   8.4333  25.5875   9.8417   8.1375  10.1708
 211.3375  57.      13.4167   7.7417   9.4833   7.7375   8.3625  23.45
  25.9292   8.6833   8.5167   7.8875  37.0042   6.45     6.95     8.3
   6.4375  39.4     14.1083  13.8583  50.4958   5.       9.8458  10.5167]
['S' 'C' 'Q' nan]
```

```
['Third', 'First', 'Second']
Categories (3, object): ['First', 'Second', 'Third']
['man' 'woman' 'child']
[ True False]
[NaN, 'C', 'E', 'G', 'D', 'A', 'B', 'F']
Categories (7, object): ['A', 'B', 'C', 'D', 'E', 'F', 'G']
['Southampton' 'Cherbourg' 'Queenstown' nan]
['no' 'yes']
[False  True]
```

Chapter 3

In [ ]:
```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
```

In [ ]:
```
# loading iris dataset through seaborn library in iris variable
iris = sns.load_dataset('iris')
# viewing first five rows
iris.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
# mean of columns of iris
iris.mean()
```

```
C:\Users\Abdullah Cheema\AppData\Local\Temp\ipykernel_3504\2237380732.py:2: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Selec
t only valid columns before calling the reduction.
  iris.mean()
```

Out[ ]:
```
sepal_length    5.843333
sepal_width     3.057333
petal_length    3.758000
petal_width     1.199333
dtype: float64
```

In [ ]:
```
# median of columns of iris
iris.median()
```

```
C:\Users\Abdullah Cheema\AppData\Local\Temp\ipykernel_3504\1858014424.py:2: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Selec
t only valid columns before calling the reduction.
  iris.median()
```

Out[ ]:
```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

In [ ]:
```
# mode of columns of iris
iris.mode()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.0	3.0	1.4	0.2	setosa
1	NaN	NaN	1.5	NaN	versicolor
2	NaN	NaN	NaN	NaN	virginica

Chapter 4 - Machine Learning Workshop

Linear Regression

- Simple linear regression is an approach for predicting a response using a single feature.
- It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

In [ ]:
```
# importing libaries
import pandas as pd
```

```python
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```python
# reading csv file "mldata2.csv" into df variable
df = pd.read_csv("Datasets/mldata2.csv")
# viewing first 5 rows
df.head()
```

Out[ ]:

	age	weight	gender	likeness	height
0	27	76.0	Male	Biryani	170.688
1	41	70.0	Male	Biryani	165
2	29	80.0	Male	Biryani	171
3	27	102.0	Male	Biryani	173
4	29	67.0	Male	Biryani	164

In [ ]:
```python
X = df.loc[:,'age'].values.reshape(-1,1) #get a copy of dataset exclude last column
y = df.loc[:,'weight'].values.reshape(-1,1) #get array of dataset in column 1st
```

In [ ]:
```python
# splitting datasets into training and testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

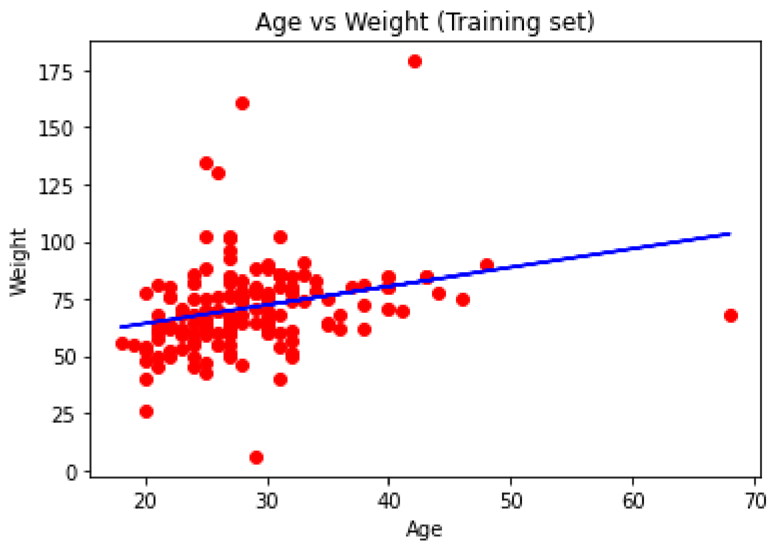
In [ ]:
```python
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

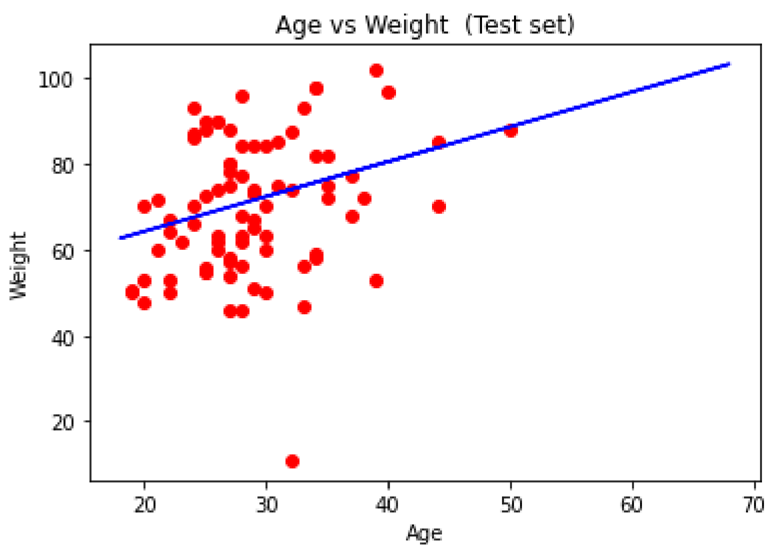
Out[ ]:
```
LinearRegression()
```

In [ ]:
```python
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Age vs Weight (Training set)')
viz_train.xlabel('Age')
viz_train.ylabel('Weight')
viz_train.show()

# Visualizing the Test set results
```

```python
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Age vs Weight  (Test set)')
viz_test.xlabel('Age')
viz_test.ylabel('Weight')
viz_test.show()
```





In [ ]:

```python
regressor.predict([[35]])
```

Out[ ]:
```
array([[76.4352169]])
```

Chapter 5 - Data Wrangling

Steps:

1. Handle Missing Values
2. Data Formatting
3. Data Normalization
   A. Scaling
   B. Centralization
4. Data Bining
   A. for groups of data
5. Making dumies of categorical data
   A. Categorical ----> Numerical

In [ ]:
```
# importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
```

In [ ]:
```
# Loading titanic dataset from seaborn library and storing in a variable ks
ks = sns.load_dataset("titanic")
# viewing first 5 rows
ks.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [ ]:
```
# storing and copying ks dataframe into ks1 and ks2
```

```
ks1 = ks.copy()
ks2 = ks.copy()
ks5 = ks.copy()
```

In [ ]:
```
# simple operation (Math operator)
ks['age']+10
```

Out[ ]:
```
0      32.0
1      48.0
2      36.0
3      45.0
4      45.0
       ...
886    37.0
887    29.0
888     NaN
889    36.0
890    42.0
Name: age, Length: 891, dtype: float64
```

1. Dealing with Missing Values

- In a dataset missing values are either Nan, N/A, 0 or a blank cell.
- Jab kabhi data na ho kisi 1 row mayy kisi 1 parameter ka (urdu)

> Steps:

1. Try to collect data if there was a mistake (Koshish karen data collect kr lein ya dekh lein agr khi galti h)
2. If missing value column does not have any effect on data then simply remove it. (Missing values wala variable (column) hi nikal dein agr data pr effect nhi hota ya simple row or data entry remove kr dein)
3. Replace the missing values
   A. How?
      a. Average value of entire variable or similar data points
      b. frequency or Mode replacement
      c. Replace based on other functions (Data Sampler knows that)
      d. Machine learning algorithms can be used
      e. Leave like that
   B. Why?
      a. Its better because no data is lost.

b. Less accurate

In [ ]:
```
# checking null values in all columns
ks.isnull().sum(axis=0)
```

Out[ ]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

We have **missing values** in **[Age, Deck, Embark Town]**

In [ ]:
```
# checking shape ( rows and columns)
ks.shape
```

Out[ ]:
```
(891, 15)
```

In [ ]:
```
# droping null values in deck with repect to all rows
ks.dropna(subset=['deck'],axis=0,inplace=True)
```

In [ ]:
```
# shape
ks.shape
```

Out[ ]:
```
(203, 15)
```

In [ ]:
```
# checking null values
ks.isnull().sum()
```

Out[ ]:
```
survived        0
pclass          0
sex             0
age            19
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck            0
embark_town     2
alive           0
alone           0
dtype: int64
```

Now we do not have any missing values in **deck** column and rows are dropped to 203

In [ ]:
```python
# if we want to drop all missing values
ks.dropna(inplace=True)
```

In [ ]:
```python
ks.isnull().sum()
```

Out[ ]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

Now we donot have any missing value.

In [ ]:
```python
ks.shape
```

Out[ ]: (182, 15)

Assignment : Replacing missing values with the average/mode of that column

In [ ]:
```
# now we replaced all null values in "age" column with the mean of its column
ks1['age'].fillna(ks1.age.mean(),inplace=True)
```

In [ ]:
```
# mode of "deck" column in dataframe
ks['deck'].mode().values[0]
```

Out[ ]: 'C'

In [ ]:
```
# now we replaced all null values in "deck" column with the mode of its column
ks1['deck'].fillna(value=ks['deck'].mode().values[0],inplace=True)
```

In [ ]:
```
ks1.isnull().sum()
```

Out[ ]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

There is no null value remaining.

2. Data Formatting/Standardization

- Standardize teh data (Data ko 1 common standard pe lana)
- Ensures data is consistent and understandable

- Easy to gather
- Easy to work with
  - Faisalabad (FSD)
  - Lahore (LHR)
  - Islamabad (ISB)
  - Karachi (KCH)
  - Peshawar (PEW)
  - Quetta (QUE)
  - Convert g to kg or similar uni for all
  - one standard unit in each column
  - ft != cm

In [ ]:

```
ks1.dtypes
```

Out[ ]:

```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
dtype: object
```

In [ ]:

```
# by astype() you can change datatype of any column
ks1['age'].astype('float64').round(1)
```

Out[ ]:
```
1      38.0
3      35.0
6      54.0
10      4.0
11     58.0
       ...
871    47.0
872    33.0
879    56.0
887    19.0
889    26.0
Name: age, Length: 182, dtype: float64
```

In [ ]:
```
# renaming the column "age" to "age_in_days"
ks1.rename(columns={'age':'age_in_days'},inplace=True)
```

In [ ]:
```
ks1.head()
```

Out[ ]:

	survived	pclass	sex	age_in_days	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True

Assignment

In [ ]:
```
# converting age in years to age in days
ks1['age_in_days'] = (ks1['age_in_days']*365).astype('int64')
```

In [ ]:
```
ks1.head()
```

Out[ ]:

	survived	pclass	sex	age_in_days	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	13870	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	12775	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	19710	0	0	51.8625	S	First	man	True	E	Southampton	no	True
10	1	3	female	1460	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False
11	1	1	female	21170	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True

3. Data Normalization

- Uniform the data
- they have the same impact
- 1 machli samundar mayy aur 1 jar mein
- Also for computational analysis

In [ ]:

```
# storing ['age_in_days','fare'] in new variable named ks3 and ks4
ks3 = ks1[['age_in_days','fare']].copy()
ks4 = ks1[['age_in_days','fare']].copy()
# viewing first 5 rows
ks3.head()
```

Out[ ]:

	age_in_days	fare
1	13870	71.2833
3	12775	53.1000
6	19710	51.8625
10	1460	16.7000
11	21170	26.5500

- The above data is really in wide range and we need to normalize and hard to compare
- Normalization change the values to the range 0-to-1 (Now both variable has the same influence in our model) ### **Methods of normalization**

1. Simple feature scaling
    - x(new)=x(old)/x(max)
2. Min-Max method
3. Z-score(standard zone) -3 to +3
4. Log transformation

In [ ]:
```
# feature scaling
ks3['age_in_days'] = ks3['age_in_days']/ks3['age_in_days'].max()
ks3['fare'] = ks3['fare']/ks3['fare'].max()
ks3.head()
```

Out[ ]:

	age_in_days	fare
1	0.4750	0.139136
3	0.4375	0.103644
6	0.6750	0.101229
10	0.0500	0.032596
11	0.7250	0.051822

In [ ]:
```
# Min max method
(ks4['fare']-ks4['fare'].min())/(ks4['fare'].max()-ks4['fare'].min())
```

Out[ ]:
```
1      0.139136
3      0.103644
6      0.101229
10     0.032596
11     0.051822
         ...
871    0.102579
872    0.009759
879    0.162314
887    0.058556
889    0.058556
Name: fare, Length: 182, dtype: float64
```

In [ ]:
```
# z-score (standard score)
(ks4['fare']-ks4['fare'].mean()) / (ks4['fare'].std())
```

Out[ ]:
```
1      -0.099835
3      -0.337554
6      -0.353732
10     -0.813428
11     -0.684654
          ...
871    -0.344689
872    -0.966388
879     0.055413
887    -0.639551
889    -0.639551
Name: fare, Length: 182, dtype: float64
```

In [ ]:
```
# checking the values was actually greater than 2
(((ks4['fare']-ks4['fare'].mean()) / ks4['fare'].std())>2).value_counts()
```

Out[ ]:
```
False    172
True      10
Name: fare, dtype: int64
```

In [ ]:
```
# log transformation
np.log(ks4['fare'])
```

```
C:\Users\Abdullah Cheema\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas\core\arraylike.py:397: Runtime
Warning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[ ]:
```
1      4.266662
3      3.972177
6      3.948596
10     2.815409
11     3.279030
         ...
871    3.961845
872    1.609438
879    4.420746
887    3.401197
889    3.401197
Name: fare, Length: 182, dtype: float64
```

4. Binning

- Grouping of values into smaller number of values
- Convert numeric into categorical ('jawan','bachay','bhooray') or 1-16,17-30 etc
- To have better understanding of groups

- low vs mid vs high

In [ ]:
```
# by age categorizing into ['Bachay','Jawan','Bhooray'] in "Categories" column
bins=[1,20,40,100]
age_groups = ['Bachay','Jawan','Bhooray']
ks5['Categories'] = pd.cut(ks5['age'],bins=bins,labels=age_groups)
```

In [ ]:
```
# viewing first 5 rows and having a Categories column at the end
ks5.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	Categorie
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	Jawa
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	Jawa
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True	Bhoora
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False	Bacha
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True	Bhoora

5. Making Dummies of Categorical Data

In [ ]:
```
# dummy variables
pd.get_dummies(ks5['sex'])
```

Out[ ]:

	female	male
1	1	0
3	1	0
6	0	1
10	1	0
11	1	0
...	...	...
871	1	0
872	0	1
879	1	0
887	1	0
889	0	1

182 rows × 2 columns

In [ ]:

```
# concat columns at the end
pd.concat([ks2, pd.get_dummies(ks2['sex'])], axis=1)
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	female
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	1
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	1
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True	0
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False	1
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
871	1	1	female	47.0	1	1	52.5542	S	First	woman	False	D	Southampton	yes	False	1
872	0	1	male	33.0	0	0	5.0000	S	First	man	True	B	Southampton	no	True	0
879	1	1	female	56.0	0	1	83.1583	C	First	woman	False	C	Cherbourg	yes	False	1
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True	1
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True	0

182 rows × 17 columns

Assignment

In [ ]:

```
# first droping sex columns and then concating columns at the end
pd.concat([ks2.drop(axis="columns",columns=['sex']), pd.get_dummies(ks2['sex'])], axis=1)
```

Out[ ]:

	survived	pclass	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	female	male
1	1	1	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	1	0
3	1	1	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	1	0
6	0	1	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True	0	1
10	1	3	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False	1	0
11	1	1	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
871	1	1	47.0	1	1	52.5542	S	First	woman	False	D	Southampton	yes	False	1	0
872	0	1	33.0	0	0	5.0000	S	First	man	True	B	Southampton	no	True	0	1
879	1	1	56.0	0	1	83.1583	C	First	woman	False	C	Cherbourg	yes	False	1	0
887	1	1	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True	1	0
889	1	1	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True	0	1

182 rows × 16 columns

Chapter 6 - Statistics in pyhton

In [ ]:

```
# importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
```

In [ ]:

```
# loading "titanic" dataset from seaborn library in ks variable
ks = sns.load_dataset('titanic')
# viewing first 5 rows
ks.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

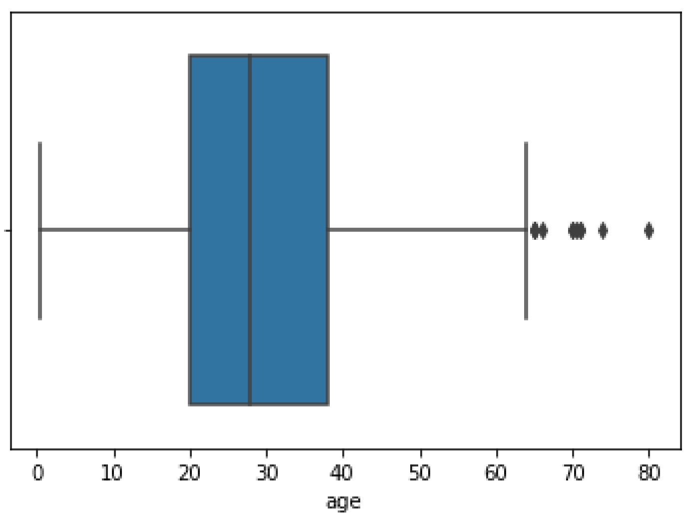
In [ ]:

```python
# box plot
sns.boxplot(ks['age'])
```

```
C:\Users\Abdullah Cheema\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `da
ta`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[ ]:

```
<AxesSubplot:xlabel='age'>
```

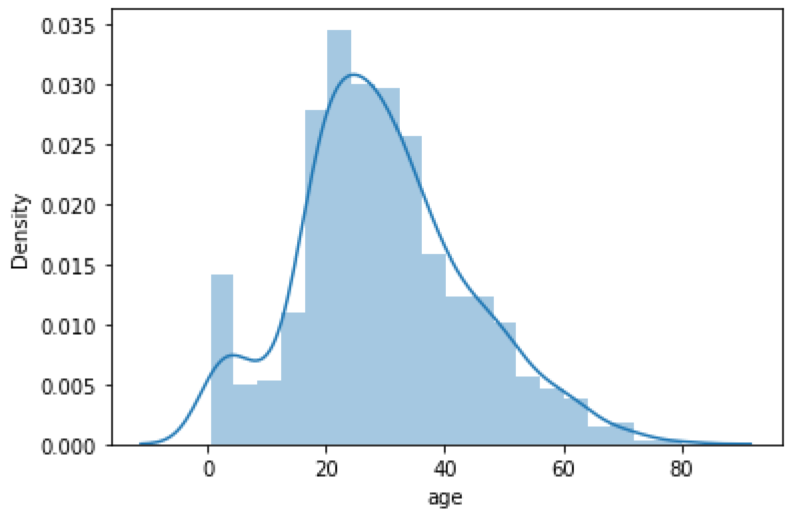


In [ ]:

```python
# distribution plot
sns.distplot(ks['age'])
```

```
C:\Users\Abdullah Cheema\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distributions.py:2619: Future
Warning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]: <AxesSubplot:xlabel='age', ylabel='Density'>



Shapiro Wilk Test

- Tests whether a data sample has Gaussian/Normal Distribution.\ Assumptions:
    - Observartion in each sample are independent and identically distributed (iid).
    - Interpretation
- H0 : The sample has a Gaussian/Normal distribution
- H1 : The sample does not have a Gaussian/Normal Distribution

In [ ]:
```python
# importing shapiro from scipy.stats library
from scipy.stats import shapiro
# droping null values
ks = ks.dropna()
stat, p = shapiro(ks['age'])
print("p = {} and stat = {}".format(p,stat))
if p>0.05:
    print('Data is normal')
else:
    print('Data is not normal')
```

```
p = 0.28414419293403625 and stat = 0.9906661510467529
Data is normal
```

1 sample tTEST

- It is a parametric test used for numerical columns
- It is used to compare mean/median etc of a column with a hypothesized mean

In [ ]:
```
from scipy import stats as st
from bioinfokit.analys import get_data
# load dataset as pandas dataframe
df = get_data('t_one_samp').data
df.head(2)
```

```
C:\Users\Abdullah Cheema\AppData\Local\Programs\Python\Python39\lib\site-packages\statsmodels\compat\pandas.py:65: Futu
reWarning: pandas.Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with t
he appropriate dtype instead.
  from pandas import Int64Index as NumericIndex
```

Out[ ]:

	size
0	5.739987
1	5.254042

In [ ]:
```
# t test using scipy
a =  df['size'].to_numpy()
# use parameter "alternative" for two-sided or one-sided test
st.ttest_1samp(a=a, popmean=5,alternative="two-sided")
```

Out[ ]:
```
Ttest_1sampResult(statistic=0.36789006583267403, pvalue=0.714539654336473)
```

Unpaired tTest or Independent tTest

- It is a parametric test used for numerical columns
- Means of two independent groups are compared.

In [ ]:
```
from scipy import stats as st
from bioinfokit.analys import get_data
# load dataset as pandas dataframe
df = get_data('t_ind_samp').data
df.head(2)
```

Out[ ]:

	Genotype	yield
0	A	78.0
1	A	84.3

In [ ]:

```
# filtering yield with genotype 'A' in a variable
a = df.loc[df["Genotype"] == 'A',"yield"].to_numpy()
# filtering yield with genotype 'B' in B variable
b = df.loc[df["Genotype"] == 'B',"yield"].to_numpy()
```

In [ ]:

```
# performing independence or paired tTest
st.ttest_ind(a=a,b=b,equal_var=True)
```

Out[ ]:

```
Ttest_indResult(statistic=-5.407091104196024, pvalue=0.00029840786595462836)
```

Paired or Dependent tTest

- It is a parametric test used for numerical columns
- Differences between the pair of dependent variables are compared.

In [ ]:

```
from bioinfokit.analys import get_data,stat
# Importing data
df = get_data('t_pair').data
# viewing first 2 rows
df.head(2)
```

Out[ ]:

	BF	AF
0	44.41	47.99
1	46.29	56.64

In [ ]:

```
from scipy import stats as st
# performing paired or dependent tTest
st.ttest_rel(a=df.AF,b=df.BF)
```

Out[ ]:

```
Ttest_relResult(statistic=14.217347189987418, pvalue=1.775932404304854e-21)
```

In [ ]:

```
# or this way
```

```
res = stat()
res.ttest(df=df,res=['AF','BF'],test_type=3,)
print(res.summary)
```

```
Paired t-test

------------------  ------------
Sample size         65
Difference Mean      5.55262
t                   14.2173
Df                  64
P-value (one-tail)   8.87966e-22
P-value (two-tail)   1.77593e-21
Lower 95.0%          4.7724
Upper 95.0%          6.33283
------------------  ------------
```

Chi squared Test

- It is a **non-parametric test** that is performed on categorical (nominal or ordinal) data.
- It helps you to understand relationship between two categorical variables eg **"smoker and and sex"**.
- It involves the frequency of events.
- It helps us to compare that we actually observed with what we expected oftentimes using population or theoratical data.
- It assists us in determining the role of random chance variation between our categorical variables.

- H0 : There is no relationship between 2 categorical variables
- H1 : There is relationship between 2 categorical variables

In [ ]:

```
# importing libraries
import seaborn as sns
import pandas as pd
import scipy.stats as stats
import numpy as np
# loading datset in df2 variable
df2 = sns.load_dataset('tips')
# viewing first 5 rows
df2.head()
```

Out[ ]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [ ]:

```
# finding a relationship between sex and smoker columns
dataset_table = pd.crosstab(index=df2.smoker,columns=df2.sex)
# frequency table
dataset_table
```

Out[ ]:

sex	Male	Female
smoker		
Yes	60	33
No	97	54

In [ ]:

```
# storing dataset_table values in variable ov
ov=dataset_table.values
print("Observed values :\n",ov)
```

```
Observed values :
 [[60 33]
 [97 54]]
```

In [ ]:

```
# contigency table
stats.chi2_contingency(dataset_table)
```

Out[ ]:

```
(0.0,
 1.0,
 1,
 array([[59.84016393, 33.15983607],
        [97.15983607, 53.84016393]]))
```

In [ ]:

```
# storing expected values in ev variable
ev = stats.chi2_contingency(dataset_table)[3]
ev
```

Out[ ]:

```
array([[59.84016393, 33.15983607],
       [97.15983607, 53.84016393]])
```

In [ ]:

```python
# finding degree of freedom
no_of_rows = len(dataset_table.iloc[0:2,0])
no_of_cols = len(dataset_table.iloc[0,0:2])
# formula for degree of freedom in Chi Squared test
ddof = (no_of_rows-1)*(no_of_cols-1)
print('Degree of Freedom',ddof)
alpha = 0.05
```

```
Degree of Freedom 1
```

In [ ]:

```python
# chi square statistic formula
chi_square = sum([(o-e)**2/e for o,e in zip(ov,ev)])
chi_square_statistic = chi_square[0]+chi_square[1]
print("Chi Square Statistic:",chi_square_statistic)
```

```
Chi Square Statistic: 0.001934818536627623
```

In [ ]:

```python
# finding critical formula
from scipy.stats import chi2
critical_value = chi2.ppf(q=1-alpha,df=ddof)
print("Critical Value",critical_value)
```

```
Critical Value 3.841458820694124
```

In [ ]:

```python
#p-value
p_value=1-chi2.cdf(x=chi_square_statistic,df=ddof)
print('p-value:',p_value)
print('Significance level: ',alpha)
print('Degree of Freedom: ',ddof)
```

```
p-value: 0.964915107315732
Significance level:  0.05
Degree of Freedom:  1
```

In [ ]:

```python
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")
```

```
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
```

We have found by using chi_square_statistic and p_value both are retaining null hypothesis that **"There is no relationship between 2 categorical variables"**